Problem 1

A simple programming exercise: Write the quintuples for a Turing Machine that transforms a unary string into its binary representation. Initially the input tape contains the number $n$ in unary notation (alphabet: \{1\}). When the machine stops, the input has been erased, and the binary representation of $n$ (alphabet \{0, 1\}) is on the tape. To simplify matters, assume that the tape is unbounded in both directions, and that the final binary representation is written to the left of the original input. If the initial tape configuration contains the number 13:

```
-----------------------------1111111111111-----------------------------
```

then when the machine stops the tape looks as follows:

```
-----------------------------1101X--------------------------------
```

Use the standard algorithm: the remainder of successive divisions by 2 produces the digits from least significant to most significant. The tape alphabet includes a separator $X$, feel free to add more characters if it helps. Try to minimize the number of states you use. It might be helpful to draw the machine as a graph, where each edge is labelled with the triple:

\[
\text{[input symbol} \rightarrow \text{output symbol, direction].}
\]

**Solution:** See next page!
PS5-2
Problem 2

From text: 3.17. Show as convincingly as you can that if a Turing Machine cannot write on the portion of the tape that contains the input, it can only recognize regular languages.

Solution:
Intuitively, the reason why a Turing Machine deprived of the capability to write on the input portion of its tape can only recognize regular languages is that, even though it could write on the blank portion of the tape, it cannot properly correlate what it is about to write with the content of the tape. In fact, once the head of the machine is moved to either the left or the right extreme of the input, it won’t be able to return to the previous position (whereas a regular Turing Machine can do so, e.g. by marking the point in the input where the head is before moving to the blank portion). Thus, writing on the blank portion of the tape is useless, and the Turing Machine is basically reduced to a 2-way Finite State Automaton (i.e. a FA that can move both left and right). To see why the ability to go back and forth on the tape does not make a 2-way Finite State Automaton more powerful than a standard Finite State Automaton, notice that one could remove transitions that cause the automaton to move left (i.e. backwards on the input) by exploiting non-determinism and possibly increasing the number of states. Removing such transitions one by one would eventually lead to a read-only Turing Machine that only moves its head forward, i.e. a standard Finite State Automaton.

One possible way to turn the above informal argument into a proof could be as follow:

1. Show that a Turing Machine $M$ that cannot write on the input portion of its tape can be simulated by a Turing Machine $N$ that does not write to its tape at all (not even in the blank part of the tape);

2. Show that the language $L = L(N)$ accepted by a Turing Machine $N$ that never writes on its tape is regular, by using Myhill-Nerode’s Theorem (a variant of which is discussed in the text as exercises 1.34 and 1.35).
Problem 3

From text: 4.15. Show that the problem of determining whether a given DFA accepts a palindrome string is decidable. Hint: consider the DFA that recognizes the reverse language. Recall also that regular languages are closed under intersection. Your proof should be an informal description of an algorithm or Turing Machine that starts from the encoding of the original DFA.

Solution:
First of all, remember that given a DFA $M_1$ that accepts a language $L_1 = \mathcal{L}(M_1)$, we can construct a DFA $M_2$ that accepts the reverse language $L_2 = \{w^R \mid w \in L_1\}$ proceeding as follows:

- reverse the direction of all the transitions in $M_1$;
- create a new node as start state, and add $\epsilon$-transitions from this new state to all the accepting states of $M_1$;
- the start state of $M_1$ becomes the new final state;
- transform the resulting NFA into an equivalent DFA proceeding as described in Theorem 1.19 in Sipser’s book.

To prove Problem 3, we use Theorem 4.4 from Sipser’s book. Using the above construction, we first convert the DFA $M_1$ into a DFA $M_2$ that accepts the reverse language. Then, we construct a new DFA $M_3$ from $M_1$ and $M_2$ that accepts only those strings that are accepted by both $M_1$ and $M_2$, i.e.:

$$\mathcal{L}(M_3) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2).$$

In other words, $M_3$ accepts only those strings in $L_1$ that are palindromes.

Notice that we can effectively build $M_3$ from $M_1$ and $M_2$ with the construction for proving the class of regular languages is closed under intersection. Once we have $M_3$ (or rather an encoding $\langle M_3 \rangle$ of $M_3$), we can use Theorem 4.4 to construct a Turing Machine $T$ that given in input $\langle M_3 \rangle$, tests whether $\mathcal{L}(M_3)$ is empty. If $T$ accepts i.e., $\mathcal{L}(M_3) \neq \emptyset$, then there must be some string accepted by $M_1$ that is palindrome: therefore, we also accept. Otherwise, we reject. \hfill \square
Problem 4

Explain why the language: \{1^n \mid n \text{ is a prime number}\} is decidable.

Solution:
In order to show that the above language is decidable, we give an high-level description of the construction of a Turing Machine $T$ that decides it. Our Turing Machine has two tapes: one for the input, and another to store (in turns) all the numbers between 2 (represented as $1^2$) and $n - 1$ (represented as $1^{n-1}$).

The idea is to let our Turing Machine $T$ try to divide $n$ by each number between 2 and $n - 1$ in turn.

To perform each trial division, $T$ acts as follows. At the beginning, the number $1^2$ is written on the second tape, and $T$ starts scanning both tapes from left to right. If the end of the second tape is reached, but there still is input to be read on the first tape, then $T$ moves the head on the second tape all the way to the left and resumes the scanning. If the end of both tapes is reached at the same time, then the number of the second tape evenly divides $n$, so that $n$ is not prime and $T$ stops and rejects. Otherwise, i.e. the end of the input was hit when the head of the second tape had not yet finished scanning, then $T$ puts the next number on the second tape and starts the whole process again.

Eventually, if all the possible trail divisions are performed and $T$ never rejected, then $n$ is prime and $T$ accepts.

Note. To speed up the trial division steps, we could have just considered the number between 2 and $\lfloor \sqrt{n} \rfloor$. In order to do that, we should have also described the way how a Turing Machine can determine the value $\lfloor \sqrt{n} \rfloor$ reading the input $1^n$. 
Problem 5

Suppose that $L$ is recursively enumerable but not recursive. Show that if $T$ is a Turing Machine accepting $L$, there must be infinitely many input strings for which $T$ loops forever.

Solution:
By definition, if $L$ is recursive enumerable, but not recursive, it means that for any Turing Machine $T$ that recognize $L$, there exists at least one input $\bar{x} \not\in L$ such that $T$ on input $\bar{x}$ loops forever.

We want to show that there actually are infinitely many such $\bar{x}$'s. Let us proceed by contradiction, assuming that the set $S$ of all the strings $\bar{x} \not\in L$ that make $T$ loop forever is finite.

We can construct another Turing Machine $T'$ such that, on input $x$:
- if $x \in S$, $T'$ halts and accepts;
- if $x \not\in S$, $T'$ halts and rejects.

Then, we can build a Turing Machine $T''$ that decides $L$ as follows. On input $x$, $T''$ first runs $T'$ with input $x$. If $T'$ accepts, then $T''$ halts and rejects. If $T'$ rejects, then $T''$ runs $T'$ on $x$.

It is clear that adding the above preprocessing step, eliminates the possibility that $T''$ loops forever. In fact:
- if $x \in S$, then $T''$ will halt and reject as soon as $T'$ stops executing;
- if $x \not\in S$, then $T''$ will get to run $T$ which, by our assumption, will stop executing accepting or rejecting according to whether $x \in L$ or $x \not\in L$.

By definition, the existence of such Turing Machine $T''$ implies that $L$ is recursive. Since we reached a contradiction, the set $S$ of input $\bar{x} \not\in L$ that make $T$ loop forever must be infinite. $\square$